Slurm: New NREL Capabilities

HPC Operations

March 2019

Presentation by: Dan Harris

Sections

1. Slurm Functionality Overview
2. Eagle Partitions by Feature
3. Job Dependencies and Job Arrays
4. Job Steps
5. Job Monitoring and Troubleshooting

https://www.nrel.gov/hpc/training.html

Slide Conventions

- Verbatim command-line interaction:
  - "$" precedes explicit typed input from the user.
  - "↲" represents hitting "enter" or "return" after input to execute it.
  - "…" denotes text output from execution was omitted for brevity.
  - "#" precedes comments, which only provide extra information.

```
$ ssh hpc_user@eagle.nrel.gov↲
    …
Password+OTPToken: # Your input will be invisible
```

- Command-line executables in prose:
  - "The command `scontrol` is very useful."

Eagle Login Nodes

Internal		External (Requires OTP Token)	
Login	DAV	Login	DAV
eagle.hpc.nrel.gov	**eagle-dav.hpc.nrel.gov**	eagle.nrel.gov	eagle-dav.nrel.gov

Direct Hostnames	
Login	DAV
el1.hpc.nrel.gov	ed1.hpc.nrel.gov
el2.hpc.nrel.gov	ed2.hpc.nrel.gov
el3.hpc.nrel.gov	ed3.hpc.nrel.gov

Sections

1. **Slurm Overview**
2. Eagle Partitions by Feature
3. Job Dependencies and Job Arrays
4. Job Steps
5. Job Monitoring and Troubleshooting

https://www.nrel.gov/hpc/eagle-user-basics.html

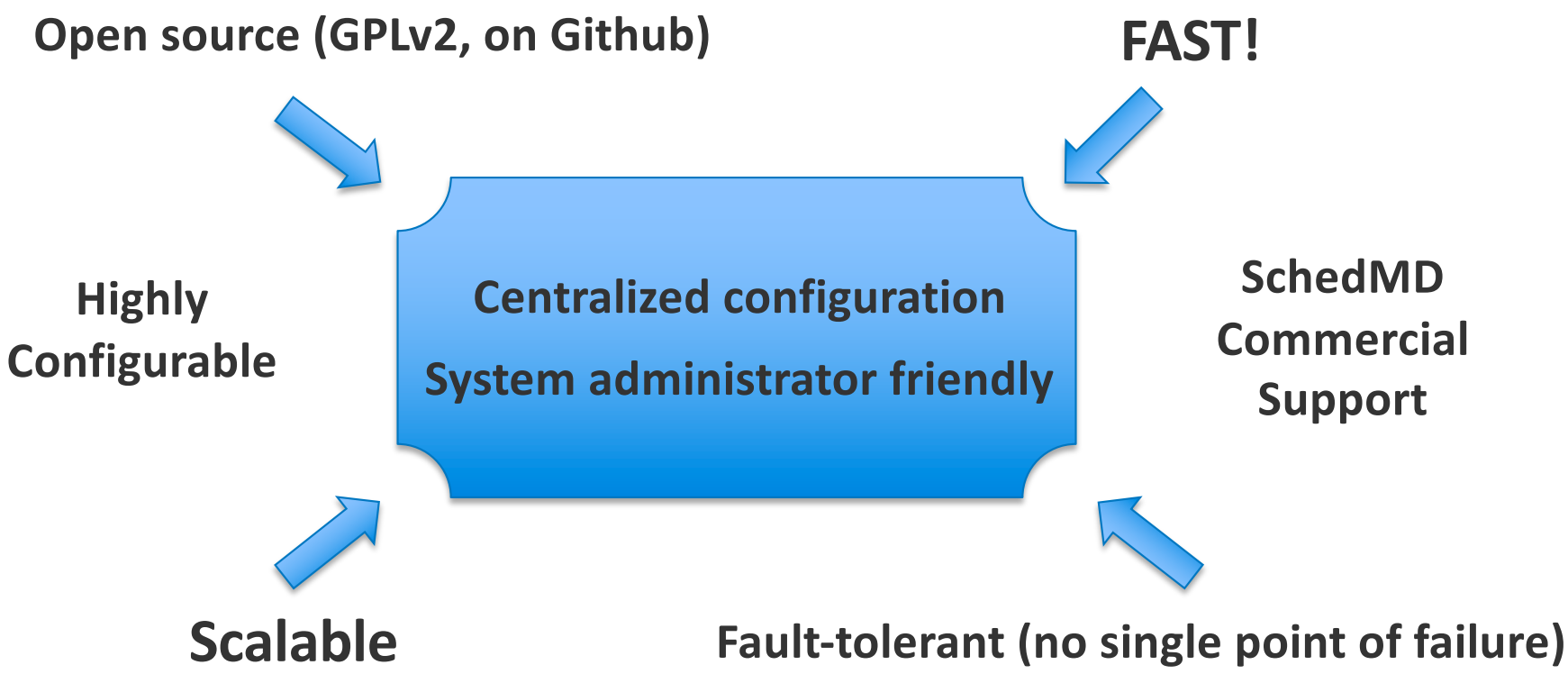

slurm
workload manager

What is Slurm

- Slurm – **S**imple **L**inux **U**tility for **R**esource **M**anagement
- Development started in 2002 at Lawrence Livermore as a resource manager for Linux clusters
- Over 500,000 lines of C code today
- Used on many of the world's largest computers
- Active global user community

https://slurm.schedmd.com/overview.html

Why Slurm?


Open source (GPLv2, on Github)
FAST!
Highly Configurable
Centralized configuration
System administrator friendly
SchedMD Commercial Support
Scalable
Fault-tolerant (no single point of failure)

Slurm Basics - Submission

- **sbatch** – Submit script to scheduler for execution
  - Script can contain some/all job options
  - Batch jobs can submit subsequent batch jobs
- **srun** - Create a job allocation (if needed) and launch a job step (typically an MPI job)
  - If invoked from within a job allocation, srun launches application on compute nodes (job step), otherwise it will create a job allocation
  - Thousands of job steps can be run serially or in parallel within a job
  - srun can use a subset of the jobs resources

Slurm Basics -Submission

- **salloc** – Create a job allocation and start shell (interactive)
  - We have identified a bug with our configuration. Your mileage may vary using salloc. Our recommended method for interactive jobs is:

```
$ srun –A <account> -t <time> [...] --pty $SHELL↲
```

- **sattach** – Connect stdin/out/err for an existing job step

Note: The job allocation commands (salloc, sbatch, and srun) accept almost identical options. There are a handful of options that only apply to a subset of these commands (e.g. batch job requeue and job array options)

Basic sbatch Example Script

```
$ cat myscript.sbatch
#!/bin/bash
#SBATCH --account=<allocation>
#SBATCH --time=4:00:00
#SBATCH --job-name=job
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=8
#SBATCH --mail-user your.email@nrel.gov
#SBATCH --mail-type BEGIN,END,FAIL
#SBATCH --output=job_output_filename.%j.out  # %j will be replaced with the job
ID

srun ./myjob.sh

$ sbatch myscript.sbatch
```

Basic srun Examples

- In our Slurm configuration, srun is preferred over mpirun
- By default, srun uses all resources of the job allocation

```
# From an interactive job:
$ srun --cpu-bind=cores my_program.sh
```

- You can also use srun to submit a job allocation
- To obtain an interactive job, you must specify a shell application as a pseudo-teletype

```
$ srun -t30 -N5 -A <handle> --pty $SHELL↲
```

Simple Linux Utility for Resource Management

- We will host more workshops dedicated to Slurm usage. Please watch for announcements, as well as our training page: https://www.nrel.gov/hpc/training.html
- We have drafted extensive and concise documentation about effective Slurm usage on Eagle: https://www.nrel.gov/hpc/eagle-running-jobs.html
- See all NREL HPC Workshop content on NREL Github: https://www.github.com/NREL/HPC

Sections

https://www.nrel.gov/hpc/eagle-job-partitions-scheduling.html

Eagle Hardware Capabilities

- Eagle comes with additional available hardware
  - All nodes have local disk space (1TB SATA) except:
    - 78 nodes have 1.6TB SSD
    - 20 nodes have 25.6TB SSD (bigscratch)
  - The standard nodes (1728) have 96GB RAM
    - 288 nodes have 192GB RAM
    - 78 nodes have 768GB RAM (bigmem)
  - 50 bigmem nodes include Dual NVIDIA Tesla V100 PCIe 16GB Computational Accelerators

Eagle Partitions

There are a number of ways to see the Eagle partitions. You can use scontrol to see detailed information about partitions

```
$ scontrol show partition
```

You can also customize the output of sinfo:

```
$ sinfo -o "%10P %.5a %.13l %.16F"
PARTITION  AVAIL     TIMELIMIT   NODES(A/I/O/T)
short         up       4:00:00   2070/4/13/2087
standard      up    2-00:00:00   2070/4/13/2087
long          up   10-00:00:00   2070/4/13/2087
bigmem        up    2-00:00:00        74/0/4/78
gpu           up    2-00:00:00       32/10/0/42
bigscratch    up    2-00:00:00       10/10/0/20
debug         up    1-00:00:00        0/13/0/13
```

Job Submission Recommendations

To access specific hardware, we strongly encourage requesting by feature instead of specifying the corresponding partition:

```
# Request 4 "bigmem" nodes for 30 minutes interactively
$ srun -t30 -N4 -A <handle> --mem=200000 --pty $SHELL↲
```

```
# Request 8 "GPU" nodes for 1 day interactively
$ srun -t1-00 -N8 -A <handle> --gres=gpu:2 --pty $SHELL↲
```

*Slurm will pick the optimal partition (known as a "queue" on Peregrine) based your job's characteristics. In opposition to standard Peregrine practice, we suggest that users **avoid specifying partitions** on their jobs with `-p` or `--partition`.*

https://www.nrel.gov/hpc/eagle-job-partitions-scheduling.html

Resources available and how to request

Resource	# of Nodes	Request
GPU	44 nodes total 22 nodes per user 2 GPUs per node	--gres=gpu:1 --gres=gpu:2
Big Memory	78 nodes total 40 nodes per user 770 GB max per node	--mem=190000 --mem=500GB
Big Scratch	20 nodes total 10 nodes per user 24 TB max per node	--tmp=20000000 --tmp=20TB

Job Submission Recommendations cont.

For debugging purposes, there is a "*debug*" partition. Use it if you need to quickly test if your job will run on a compute node with `-p debug` or `--partition=debug`

```
$ srun -t30 -A handle -p debug --pty $SHELL↲
```

There is now a dedicated GPU partition following the convention above. Use -p gpu or --partition-gpu

There are limits to the number of nodes in these partitions. You may use shownodes to quickly view usage.

Node Availability

To check the availability of what hardware features are in use, run `shownodes`. Similarly, you can run `sinfo` for more nuanced output.

```
$ shownodes↲
partition     # free USED reserved completing offline down
------------- - ---- ---- -------- ---------- ------- ----
bigmem        m    0   46        0          0       0    0
debug         d   10    1        0          0       0    0
gpu           g    0   44        0          0       0    0
standard      s    4 1967        7          4      10   17
------------- - ---- ---- -------- ---------- ------- ----
       TOTALs     14 2058        7          4      10   17
           %s    0.7 97.5      0.3        0.2     0.5  0.8
```

Eagle Walltime

A maximum walltime is **required** on all Eagle job submissions. Job allocations will be rejected if not specified:

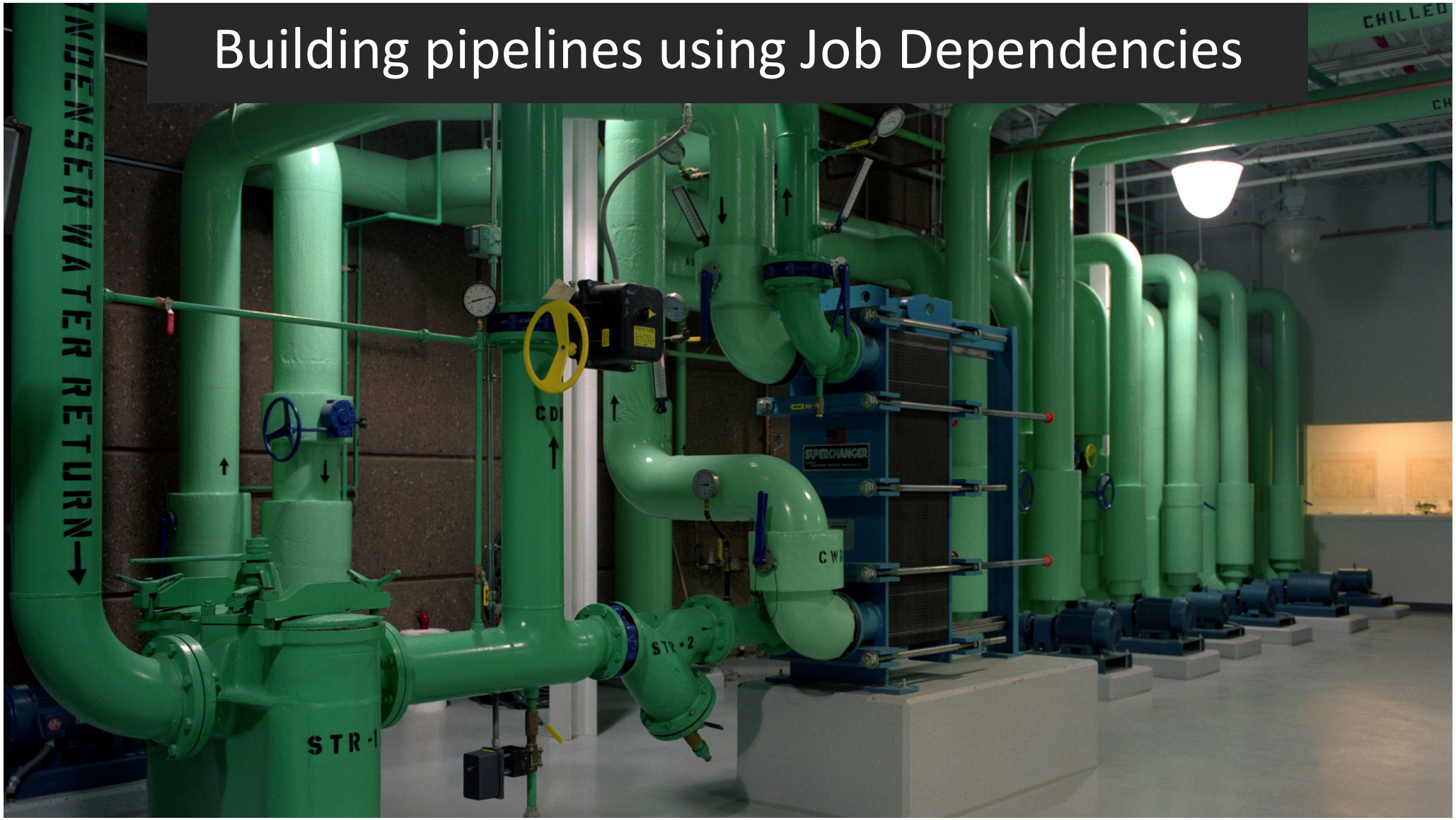
```
$ srun -A handle --pty $SHELL↲
error: Job submit/allocate failed: Time limit specification
required, but not provided
```

A **minimum** walltime may allow your job to start sooner using the backfill scheduler.

```
# 100 nodes for 2 days with a MINIMUM time of 36 hours
$ srun -t2-00 –N100 -A handle --time-min=36:00:00 --pty $SHELL↲
```

Sections

Building pipelines using Job Dependencies



Job Dependencies

- Job dependencies are used to defer the start of a job until the specified dependencies have been satisfied.
- Many jobs can share the same dependency and these jobs may even belong to different users.
- *Once a job dependency fails due to the termination state of a preceding job, the dependent job will* ***never*** *run,* ***even if*** *the preceding job is requeued and has a different termination state in a subsequent execution.*

Job Dependency Options

- **`after:job_id[:job_id...]`**
  This job can begin execution after the specified jobs have begun execution.

- **`afterany:job_id[:job_id...]`**
  This job can begin execution after the specified jobs have terminated (regardless of state.)

- **`afternotok:job_id[:job_id...]`**
  This job can begin execution after the specified jobs have terminated in some failed state.

- **`afterok:job_id[:job_id...]`**
  This job can begin execution after the specified jobs have successfully executed.

- **`aftercorr:job_id`**
  A task of this job array can begin execution after the corresponding task ID in the specified job has completed successfully.

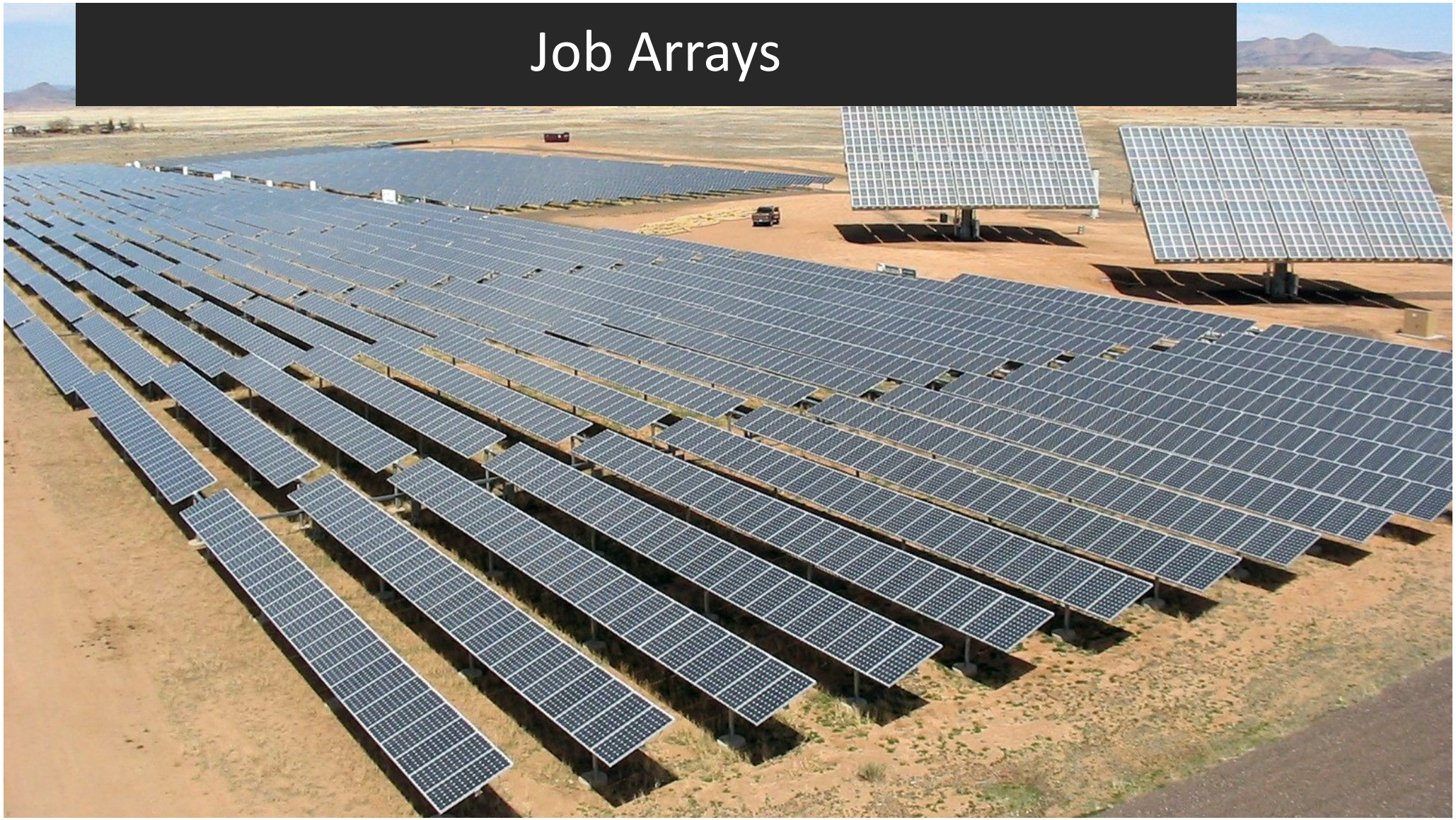
- **`singleton`**
  This job can begin execution after any previously launched jobs sharing the same job name and user have terminated.

Dependency Sequence Example

Submit sequence of batch jobs:

```
$ sbatch --ntasks=1 --time=10 pre_process.bash
Submitted batch job 1010
$ sbatch --ntasks=128 --time=60 --dependency=afterok:1010 do_work.bash
Submitted batch job 1011
$ sbatch --ntasks=1 --time=30 --dependency=afterok:1011 post_process.bash
Submitted batch job 1012
…
$ sbatch --begin=17:00:00 -n1 -t10 --dependency=afterany:1011,1012 night.bash
Submitted batch job 1013
```

Job Arrays



Job Arrays

- Job arrays are an efficient mechanism of managing a collection of batch jobs with identical resource requirements
- Most Slurm commands can manage job arrays either as individual elements (tasks) or as a single entity (e.g. delete an entire job array in a single command)
- Job Arrays are only supported using the `sbatch` command

Job Array examples

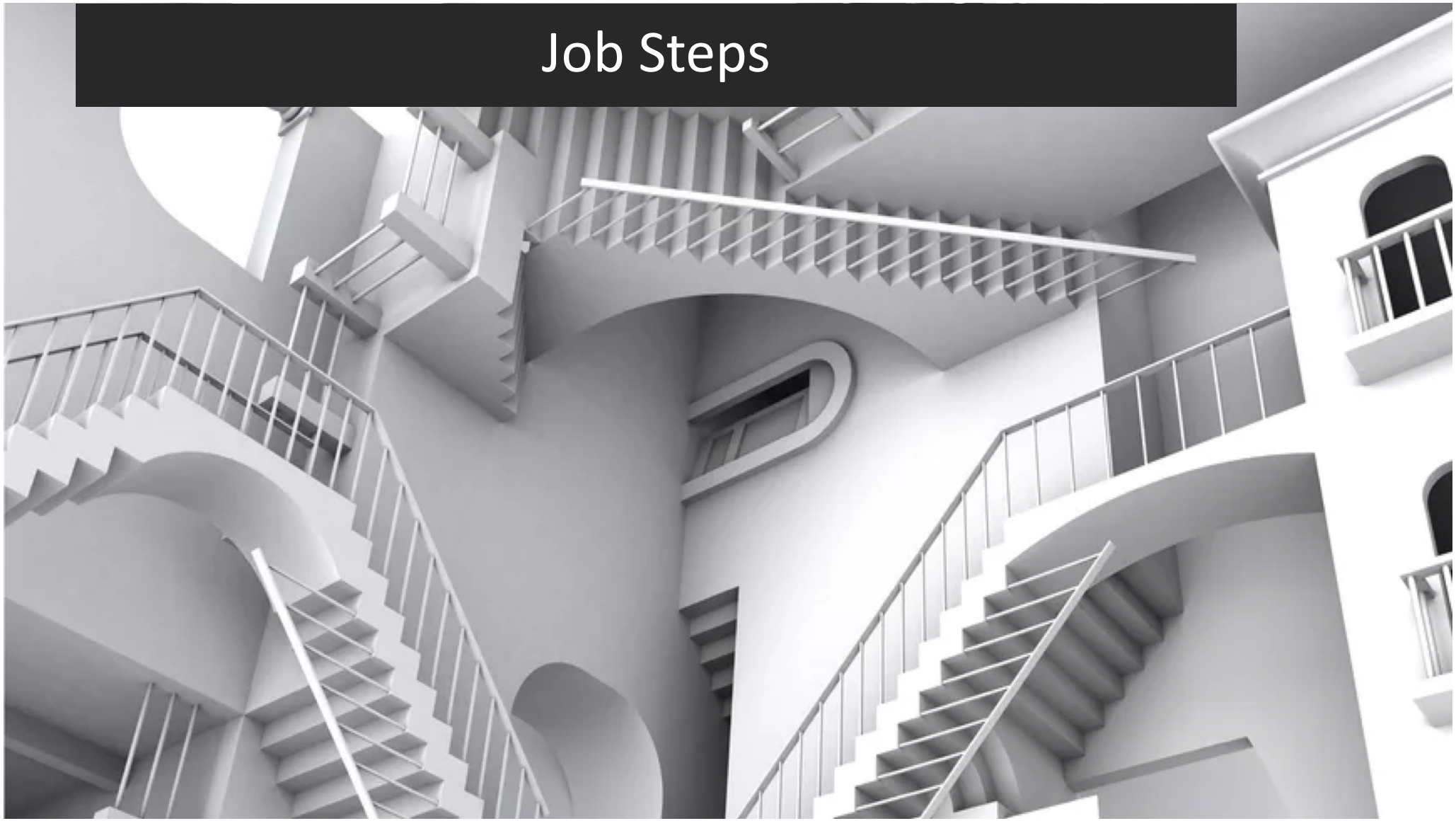
- All jobs must have the same initial options (e.g. size, time limit, etc.)
- It is possible to change some of these options after the job has begun execution using the scontrol command specifying the *JobID* of the array or individual *ArrayJobID*.

```
# Submit a job array with index values between 0 and 100
$ sbatch --array=0-100 -N1 array.sh↲

# Submit a job array with index values of 1, 3, 5 and 7
$ sbatch --array=1,3,5,7 -N1 array.sh

# Submit a job array with index values between 1 and 7 with a step size
# of 2 (i.e. 1, 3, 5 and 7)
$ sbatch --array=1-7:2 -N1 array.sh
```

Job Steps



Job Step Best practices

- Each Slurm job can contain a multitude of job steps and the overhead in Slurm for managing job steps is much lower than that of individual jobs.
- Consider putting related work into a single Slurm job with multiple job steps both for performance reasons and ease of management.
- Similarly, it is easy to map several single-core jobs to individual cores of a single job to conserve NREL hours.

Sequential Job Steps

```
#!/bin/bash
#SBATCH --account=<allocation>
#SBATCH --time=4:00:00
#SBATCH --job-name=steps
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=8
#SBATCH --output=job_output_filename.%j.out  # %j will be replaced
with the job ID

# By default, srun uses all job allocation resources (8 tasks each)
srun --cpu-bind=cores ./myjob.1a
srun --cpu-bind=cores ./myjob.1b
srun --cpu-bind=cores ./myjob.1c
```

Parallel Job Steps

```
#!/bin/bash
#SBATCH --account=<allocation>
#SBATCH --time=4:00:00
#SBATCH --job-name=steps
#SBATCH --nodes=8
#SBATCH --output=job_output_filename.%j.out

# Be sure to request enough nodes to run all job steps at the same time
srun -N 2 -n 44 -c 2 --cpu-bind=cores ./myjob.1 &
srun -N 4 -n 108 -c 2 --cpu-bind=cores ./myjob.2 &
srun -N 2 -n 40 -c 2 --cpu-bind=cores ./myjob.3 &
wait
```

Sections

1. Slurm Overview
2. Eagle Partitions by Feature
3. Job Dependencies and Job Arrays
4. Job Steps
5. **Job Monitoring and Troubleshooting**

https://www.nrel.gov/hpc/eagle-monitor-control-commands.html

Job Information

- Show queued jobs by user

```
$ squeue -u <username> -l ↲
```

- Estimate when your job(s) will start

```
$ squeue --start -j <jobID>,<jobID>↲
```

- Show detailed job information

```
$ scontrol show job <jobID>↲
```

- Show hostnames of allocated nodes

```
$ scontrol show hostname↲
```

- Write submission script to file

```
$ scontrol write batch_script <jobID>↲
```

https://www.nrel.gov/hpc/eagle-monitor-control-commands.html

Job Information continued

- Use `sacct` for job accounting information (and exit codes)
- (Use `sacct -e` for a list of `--format` options)

```
$ sacct --starttime 03/01 --format=JobID,Jobname,state,time,elapsed,ncpus,exit
       JobID    JobName      State  Timelimit    Elapsed      NCPUS ExitCode
------------ ---------- ---------- ---------- ---------- ---------- --------
605590             bash CANCELLED+   04:00:00   00:00:00        250      0:0
605591             bash  COMPLETED   04:01:00   00:00:07       9000      0:0
605591.exte+     extern  COMPLETED              00:00:07       9000      0:0
605591.0           bash  COMPLETED              00:00:04       9000      0:0
605595             bash     FAILED   04:01:00   00:00:09       2160    127:0
605595.exte+     extern  COMPLETED              00:00:09       2160      0:0
605595.0           bash     FAILED              00:00:07       2160    127:0
```

https://www.nrel.gov/hpc/eagle-monitor-control-commands.html

Update Job Allocations using scontrol

The command `scontrol` can be used to update queued and running jobs

```
$ scontrol update jobid=526501 qos=high
$ sacct -j 526501 --format=jobid,partition,state,qos
       JobID  Partition      State        QOS
------------ ---------- ---------- ----------
526501             short    RUNNING       high
526501.exte+                RUNNING
526501.0                  COMPLETED
```

To pause a job: `scontrol hold <JOBID>`
To resume a job: `scontrol resume <JOBID>`
To cancel and rerun: `scontrol requeue <JOBID>`

https://www.nrel.gov/hpc/eagle-monitor-control-commands.html

Tracking Allocation Usage

`alloc_tracker` has been deprecated.

Please use **`hours_report`** instead.

```
[hpc_user@el1 ~]$ hours_report↲
Gathering data from database.....Done
    …
User hpc_user has access to and used:
Allocation Handle    System     Hours Used Note
-------------------- ---------- ---------- ----
handle               Peregrine         125
handle               Eagle             320
```

Advanced Tracking

`hours_report --showall`

- List each project, its PI, and its NREL hour usage.

`hours_report --showall --drillbyuser`

- List each project like above, but also show each member's contributing usage of allotted hours.

`hours_report --help`

- List usage instructions. `hours_report` is still in development and new features will be documented here.

Feedback is Appreciated!

If you have any suggestions to improve this presentation we invite you to share with us at HPC-Help@nrel.gov

Thank You

www.nrel.gov

NREL is a national laboratory of the U.S. Department of Energy, Office of Energy Efficiency and Renewable Energy, operated by the Alliance for Sustainable Energy, LLC.

NREL
NATIONAL RENEWABLE ENERGY LABORATORY